Metaheuristic Development Methodology

Fall 2009

Instructor: Dr. Masoud Yaghini

Phases and Steps

Phases and Steps

- **Phase 1: Understanding Problem**
  - Step 1: State the Problem
  - Step 2: Review of Existing Solution Methods
  - Step 3: Define Goals
  - Step 4: Select Instances

Phases and Steps

- **Phase 2: Design of Algorithm**
  - Step 5: Select Solution Strategy
  - Step 6: Define Performance Measures
  - Step 7: Select Data Structures
  - Step 8: Specify Algorithm
  - Step 9: Verify Algorithm
  - Step 10: Analyze Algorithm

Phases and Steps

- **Phase 3: Implementation**
  - Step 11: Implement Algorithm
  - Step 12: Tune Parameters
  - Step 13: Analyze the Performance of Algorithm
  - Step 14: Report Results

Step 1. State the Problem

State the Problem

- This is the first step in designing of algorithm.
- In this step first of all you need to understand and state the problem completely.
- The problem statements should be very clear.
- Inputs, outputs, and assumptions of problem should be defined.
- It is better we can provide mathematical model for clarity.

Step 2. Review of Existing Solution Methods

Review of Existing Solution Methods

- There are some types of problems that are commonly occurring and to solve such problems there are typical algorithms which are already available.
- Hence if the given problem is a common type of problem, then already existing algorithms (exact or heuristic) as a solution to that problem can be used.
- After reviewing such an existing algorithm it is necessary to find its strength and weakness
  - For example, efficiency, memory utilization

Step 3. Define Goals

Define Goals

- In the development of a metaheuristic, the goals must be clearly defined.
- All the experiments, performance analysis measures, and statistical analysis will depend on the purpose of designing the metaheuristic.

Define Goals

- A contribution may be obtained for different criteria such as:
  - search time,
  - quality of solutions,
  - robustness in terms of the instances,
  - solving large-scale problems,
  - parallel scalability in terms of the number of processors,
  - easiness of implementation,
  - easiness to combine with other algorithms,
  - flexibility to solve other problems or optimization models,
  - innovation using new nature-inspired paradigms,
  - automatic tuning of parameters,
  - providing a tight approximation to the problem,
  - and so on.

Step 4. Select Instances

Select Instances

- Once the goals are defined, the selection of the input instances to perform the evaluation must be carefully done.
- The structure associated with the input instances may influence significantly the performance of metaheuristics.
- Two types of instances exist:
  - **Real-life instances**
  - **Constructed instances**

Real-life instances

- They represent practical instances of the problem to be solved.
- If available, they constitute a good benchmark to carry out the performance evaluation of a metaheuristic.
- It is difficult to obtain real-life instances
  - those data are not public
  - it is difficult to obtain a large number of real-life instances for financial reasons.
  - Also, collecting some real-life instances may be time consuming.

Constructed Instances

- Many public libraries of “standard” instances are available on Internet, such as
  - OR-Library,
    - http://people.brunel.ac.uk/~mastjjb/jeb/info.html
  - MIPLIB,
    - http://www.caam.rice.edu/~bixby/miplib/miplib.html
  - TSPLIB for the traveling salesman problem
    - http://softlib.rice.edu/softlib/tsplib/
  - DIMACS challenges
    - http://dimacs.rutgers.edu/Challenges/

Constructed Instances

- In addition to some real-life instances, those libraries contain randomly generated instances.
- A disadvantage of random instances is that they are often too far from real-life problems to reflect their structure and important characteristics.
- The advantage of constructed instances is that different instances in size and structure may be generated.

Constructed Instances

- Evaluating the performances of a given metaheuristic using only **random instances may be controversial**.
- **For instance**,
  - the structure of uniformly generated random instances may be completely different from real-life instances of the problem, and
  - then the effectiveness of the metaheuristic will be completely different in practice.

Example

- Let us consider the symmetric TSP problem with n cities where the distance matrix is generated as follows:
  - each element dij, i # j, of the distance matrix is independently generated between [0, 20] using a uniform distribution.
- Any randomly generated tour represents a good solution.
- For example, for an instance of 5000 cities, it has been shown that the standard deviation is equal to 408 ($\sigma\sqrt{n}$) and the average cost is 50, 000 ($10 \cdot n$).

Example

- Almost any tour will have a good quality (i.e., cost of ±3(408) of 50, 000).
- Hence, evaluating a metaheuristic on such instances is a pitfall to avoid.

Experimental Design

- The set of instances must be diverse in terms of:
  - the size of the instances,
  - their difficulties, and
  - their structure.
- The instances must be divided into two subsets:
  - the first subset will be used **to tune** the parameters of the metaheuristic and
  - the second subset **to evaluate** the performance of the search algorithms.

Experimental Design

- The values of the parameters associated with the used metaheuristics must be same for all instances.
- No fine-tuning of the values is done for each instance unless the use of an automatic off-line or online initialization strategy
- Indeed, this will cause an **overfitting** of the metaheuristic in solving known and specific instances.

Experimental Design

- The parameter values will be excellent to solve the instances that serve to calibrate the parameters and very poor to tackle other instances.
- The robustness of the metaheuristic will be affected to solve unknown instances.
- Otherwise, the time to determine the parameter values of the metaheuristic to solve a given instance must be taken into account in the performance evaluation.
- Different parameter values may be adapted to different structures and sizes of the instances.

Step 5. Select Solution Strategy

Select Solution Strategy

- The next important step is to decide whether the problem is to be solved exactly or approximately.
- If the problem needs to be solved correctly then we need exact algorithm.
- Otherwise if the problem is so complex that we won't get the exact solution then in that situation we need to choose approximate algorithms.

Select Solution Strategy

- We can choose one of following options:
  - an exact algorithm
  - a hybrid exact and heuristic algorithm
  - a heuristic algorithm
  - a metaheuristic algorithm
  - a hybrid metaheuristic and heuristic algorithm
  - a hybrid exact and metaheuristics algorithm
  - a hybrid metaheuristics algorithm

Select Solution Strategy

- After selecting algorithmic strategy we should choose the exact, heuristic, or metaheuristic method, such as:
  - Exact methods:
    - Simplex
    - Branch and Bound
    - Branch and cut
    - Column generation
  - Metaheuristic:
    - Genetic algorithm
    - Ant colony optimization
    - Tabu search
    - Simulated annealing

Step 6. Define the Performance Measures

Define the Performance Measures

- In this step, the performance measures and indicators to compute are selected.
- In exact optimization methods, the efficiency in terms of **search time** is the main indicator to evaluate the performances of the algorithms as they guarantee the global optimality of solutions.
- Indicators to evaluate the effectiveness of metaheuristic search methods:
  - **Quality of Solutions**
  - **Computational effort**
  - **Robustness**

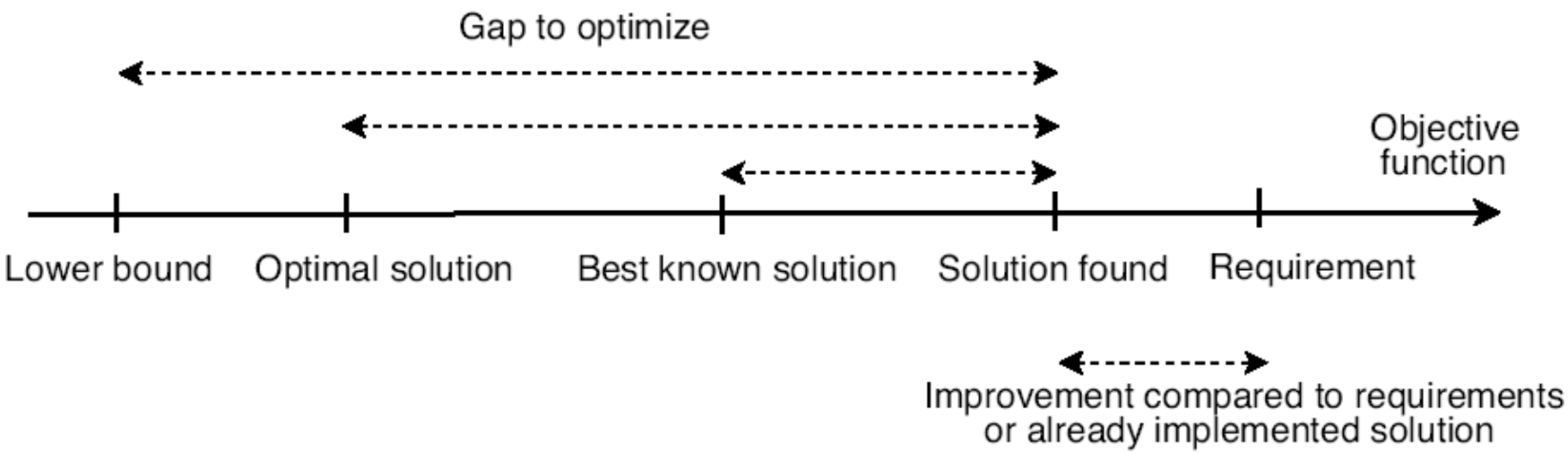
Quality of Solutions

Quality of Solutions

- Performance indicators for defining the quality of solutions in terms of precision are generally based on measuring the distance or the percent deviation of the obtained solution to one of the following solutions:
  - **Global optimal solution**
  - **Lower/upper bound solution**
  - **Best known solution**
  - **Requirements or actual implemented solution**

Quality of Solutions

- Performance assessment of the quality of the solutions. We suppose a minimization problem.



Global optimal solution

- The use of global optimal solutions allows a more absolute performance evaluation of the different metaheuristics.
- The absolute difference may be defined as
  - $|f(s) - f(s^*)|$
  - $|f(s) - f(s^*)| / f(s^*)$
  - where s is the obtained solution and s^* is the global optimal solution.

Global optimal solution

- The global optimal solution may be found by an exact algorithm or may be available using “constructed” instances where the optimal solution is known
- Unfortunately, for many complex problems, global optimal solutions could not be available.

Lower/upper bound solution

- For optimization problems where the global optimal solution is not available, tight lower bounds (minimization problem) may be considered as an alternative to global optimal solutions.
- For some optimization problems, tight lower bounds are known and easy to obtain.

Lower/upper bound solution

- **Example: Simple lower bound for the TSP:**
  - The Held–Karp (HK) 1-tree lower bound for the symmetric TSP problem is quick and easy to compute.
  - Given an instance (V, d) where V is the set of n cities and d the distance matrix.
  - A node $v_0 \in V$ is selected.
  - Let r be the total edge length of a minimum spanning tree over the n - 1 cities ($v \in V - \{v_0\}$).
  - The lower bound t is represented by the r value plus the two cheapest edges incident on v_0.

$$t = r + \min\{d(v_0, x) + d(v_0, y) : x, y \in V - \{v_0\}, x \neq y\}$$

Lower/upper bound solution

- **Example: Simple lower bound for the TSP:**
  - Indeed, any TSP tour must use two edges e and f incident on the node v_0.
  - Removing these two edges and the node v_0 from the tour yields a spanning tree of $V - \{v_0\}$.
  - Typically, the lower bound t is 10% below the global optimal solution.

Lower/upper bound solution

- Different relaxation techniques may be used to find lower bounds such as:
  - The classical continuous relaxation and
  - The Lagrangian relaxation
- In **continuous relaxation** for IP problems, the variables are supposed to be real numbers instead of integers.
- In **Lagrangian relaxation**, some constraints multiplied by **Lagrange multipliers** are incorporated into the objective function

Lower/upper bound solution

- If the gap between the obtained solution and the lower bound is small, then the distance of the obtained solution to the optimal solution is smaller

Best known solution

- For many classical problems, there exist libraries of standard instances available on the Web.
- For those instances, the best available solution is known and is updated each time an improvement is found.

Requirements or actual implemented solution

- For real-life problems, a decision maker may define a requirement on the quality of the solution to obtain.
- This solution may be the one that is currently implemented.
- These solutions may constitute the reference in terms of quality.

Computational Effort

Computational Effort

- The efficiency of a metaheuristic may be demonstrated using:
  - A theoretical analysis
  - An empirical analysis
- In **theoretical analysis**, the worst-case complexity or average-case complexity of the algorithm is generally computed.
- In **empirical analysis**, measures related to the computation time of the metaheuristic used to solve a given instance are reported.

Empirical Analysis

- The meaning of the computation time must be clearly specified:
  - CPU time
  - with or without input/output time
  - with or without preprocessing/postprocessing time
- The main drawback of computation time measure is that it depends on:
  - The computer characteristics such as the hardware (e.g., processor, memories: RAM and cache, parallel architecture),
  - Operating systems,
  - Language, and compilers on which the metaheuristic is executed.

Empirical Analysis

- Some indicators that are independent of the computer system may also be used, such as the number of objective function evaluations.
  - It is an acceptable measure for time-intensive and constant objective functions.
  - Using this metric may be problematic for problems where the evaluation cost is low compared to the rest of the metaheuristics or is not time constant in which it depends on the solution evaluated and time.

Robustness

Robustness

- There is no commonly acceptable definition of robustness.
- Different alternative definitions exist for robustness.
- In general, robustness is insensitivity against small deviations in the input instances (data) or the parameters of the metaheuristic.
- The **lower the variability** of the obtained solutions the **better the robustness**

Robustness

- The metaheuristic should be able to perform well on a large variety of instances and/or problems using the same parameters.
- In stochastic algorithms (e.g. genetic algorithm), the robustness may also be related to the average/deviation behavior of the algorithm over different runs of the algorithm on the same instance.

Step 7. Select Data Structures

Select Data Structures

- Data structure and algorithm work together and these are interdependent.
- Hence choice of proper data structure is required before designing the actual algorithm.
- The implementation of algorithm (program) is possible with the help of algorithm and data structure.

Step 8. Specify Algorithm

Specify Algorithm

- There are various ways by which we can specify an algorithm:
  - **Using natural language**
  - **Pseudo code**
  - **Flowchart**

Using Natural Language

- It is very simple to specify an algorithm using natural language.
- But many times specification of algorithm by using **natural language** is not clear.
- **For example :** Write an algorithm to perform addition of two numbers:
  - Step 1 : Read the first number say a.
  - Step 2 : Read the second number say b.
  - Step 3 : Add the two numbers and store the result in a variable c.
  - Step 4 : Display the result.

Using Natural Language

- Specification of algorithm by using natural language creates difficulty while actually implementing it.
- Hence many programmers prefer to have specification of algorithm by means of **pseudo code**.

Pseudo Code

- **Pseudo code** is a combination of natural language and programming language constructs.
- A pseudo code is usually more precise than a natural language.

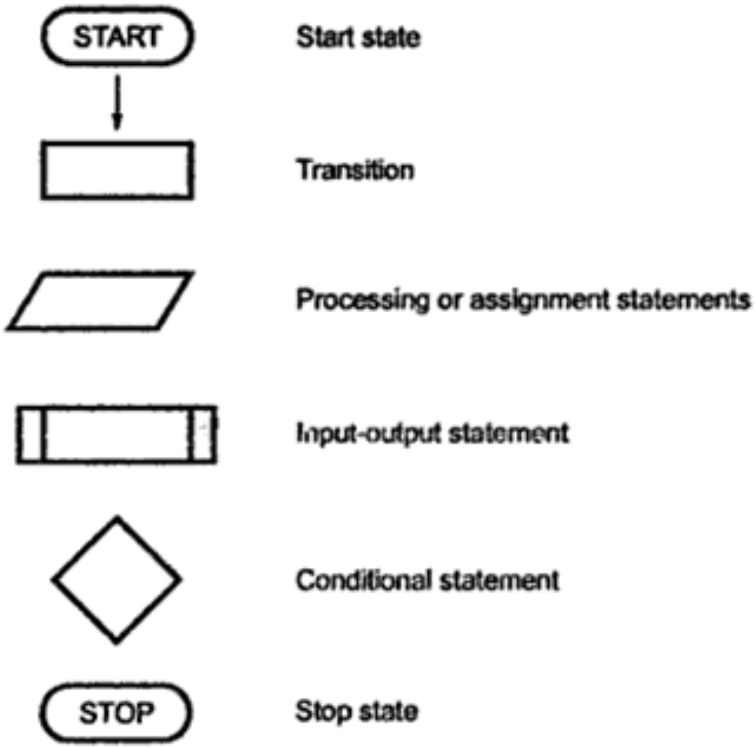
Pseudo Code

- **For example**: Write an algorithm for performing addition of two numbers.

```
Algorithm sum(a, b)
// Problem Description: This algorithm performs
   addition of two integers
// Input: two integers a and b
// Output: addition of two integers
C <- a + b
write (c)
```

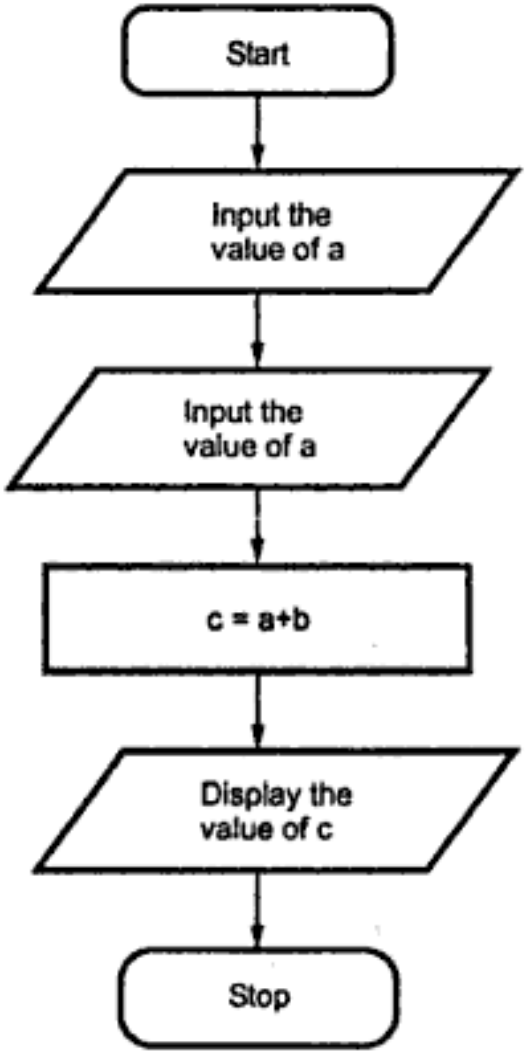
Flowchart

- Another way of representing the algorithm is by **flowchart**.
- Flowchart is a graphical representation of an algorithm.
- Typical symbols used in flowchart are:



Flowchart

- For example:



Step 9. Verify Algorithm

Verify Algorithm

- Algorithmic verification means checking correctness of an algorithm.
- After specifying an algorithm we go for checking its correctness.
- We normally check whether the algorithm gives correct output in finite amount of time for a valid set of input.
- The proof of correctness of an algorithm can be complex sometimes.
- But to show that an algorithm works incorrectly we have to show that at least for one instance of valid input the algorithm gives wrong result.

Step 10. Analyze Algorithm

Analyze Algorithm

- While analyzing an algorithm we should consider following factors:
  - Time complexity (efficiency) of an algorithm
  - Space efficiency of an algorithm
  - Simplicity of an Algorithm
  - Generality of an algorithm

Analyze Algorithm

- **Time complexity of an algorithm**
  - means the amount of time taken by an algorithm to run.
  - By computing time complexity we come to know whether the algorithm is to run slow or fast.
- **Space complexity of an algorithm**
  - means the amount of space (memory) taken by an algorithm.
  - By computing space complexity we can analyze whether an algorithm requires more or less space.

Analyze Algorithm

- **Simplicity is of an algorithm**
  - means generating sequence of instructions which are easy to understand.
  - This is an important characteristic of an algorithm because simple algorithms can be understood quickly and one can then write simpler programs for such algorithms.
  - Finding out bugs from algorithm or debugging the program becomes easy when algorithm is simple.
  - Sometimes simpler algorithms arc more efficient than complex algorithms.
  - But it is not always possible that the algorithm is simple.

Analyze Algorithm

- **Generality**
  - Becomes an algorithm in more general way rather than designing it for particular set of input.
  - Hence we should write general algorithms always.
  - For example designing an algorithm for finding greatest common divisor (GCD) of any two numbers is more appealing than that of particular two values.
  - But sometimes it is not at all required to design a generalized algorithm.

Analyze Algorithm of Algorithm

- Analysis of algorithm means checking the characteristics such as : time complexity, space complexity, simplicity, generality and range of input.
- If these factors are not satisfactory then we must redesign the algorithm.

Step 11. Implement Algorithm

Implement Algorithm

- The implementation of an algorithm is done by suitable programming language.
- For example, if an algorithm consists of objects and related methods then it will be better to implement such algorithm using some object oriented programming language like C++ or JAVA.
- While writing a program for given algorithm it is essential to write an optimized code.

Step 12. Tune Parameters

Tune Parameters

- Many parameters have to be tuned for any metaheuristic.
- The parameters may have a great influence on the efficiency and effectiveness of the search.
- It is not obvious to define a priori which parameter setting should be used.
- The optimal values for the parameters depend mainly on:
  - **the problem**
  - **the instance** of the problem to deal with
  - **the search time** that we want to spend in solving the problem

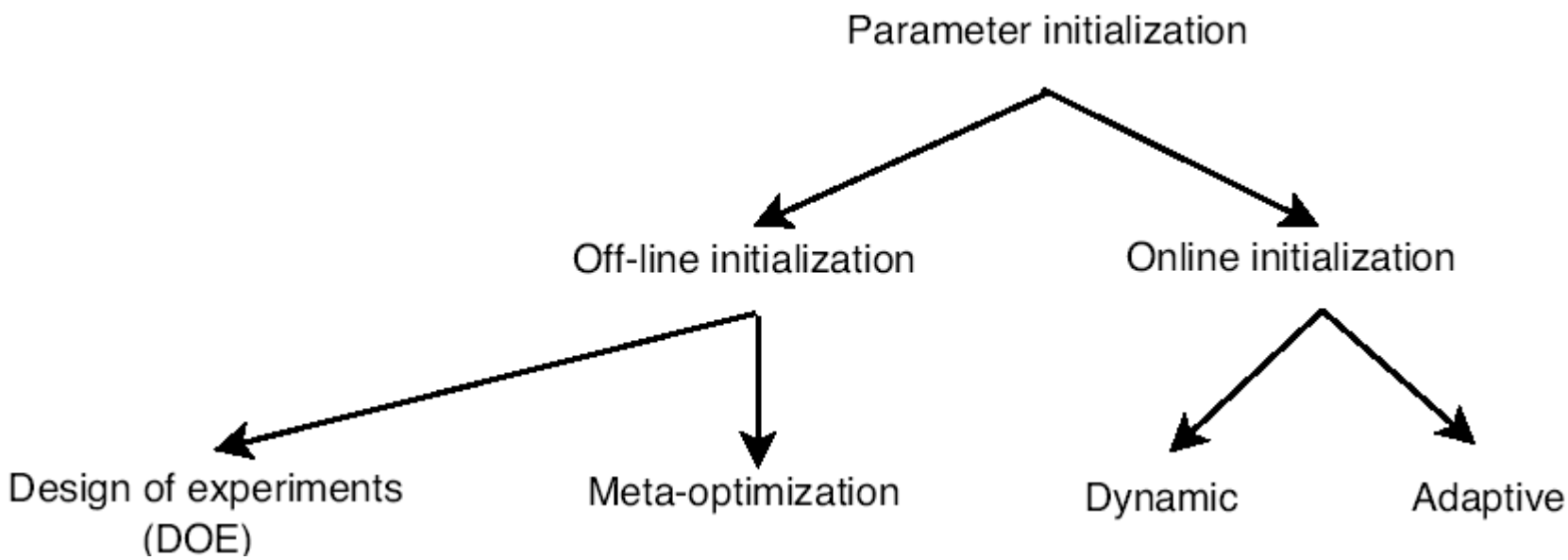
Tune Parameters

- A universally optimal parameter values set for a given metaheuristic does not exist.
- The parameters are not only numerical values but may also involve the use of search components.
- There are two different strategies for parameter tuning:
  - the **off-line parameter tuning strategy** (or meta-optimization)
  - the **online parameter tuning strategy**

Tune Parameters

- Off-line tuning strategy
  - the values of different parameters are fixed before the execution of the metaheuristic
- Online tuning strategy
  - the parameters are controlled and updated dynamically or adaptively during the execution of the metaheuristic.

Parameter Initialization Strategies



Design of Experiments (DoE)

Off-Line Parameter Initialization

- Usually, metaheuristic designers tune one parameter at a time, and its optimal value is determined empirically.
- In this case, no interaction between parameters is studied.
- This sequential optimization strategy (i.e., one-by-one parameter) do not guarantee to find the optimal setting even if an exact optimization setting is performed.

Design of Experiments (DoE)

- To overcome this problem, experimental design is used.
- Before using an experimental design approach, the following concepts must be defined:
  - Factors that represent the parameters to vary in the experiments.
  - Levels that represent the different values of the parameters, which may be quantitative (e.g., mutation probability) or qualitative (e.g., neighborhood).

Design of Experiments (DoE)

- Let's consider n factors in which each factor has k levels, a full factorial design needs n^k experiments.
- Then, the "best" levels are identified for each factor.
- This approach is its high computational cost especially when the number of parameters (factors) and their domain values are large, that is, a very large number of experiments must be realized.

Design of Experiments (DoE)

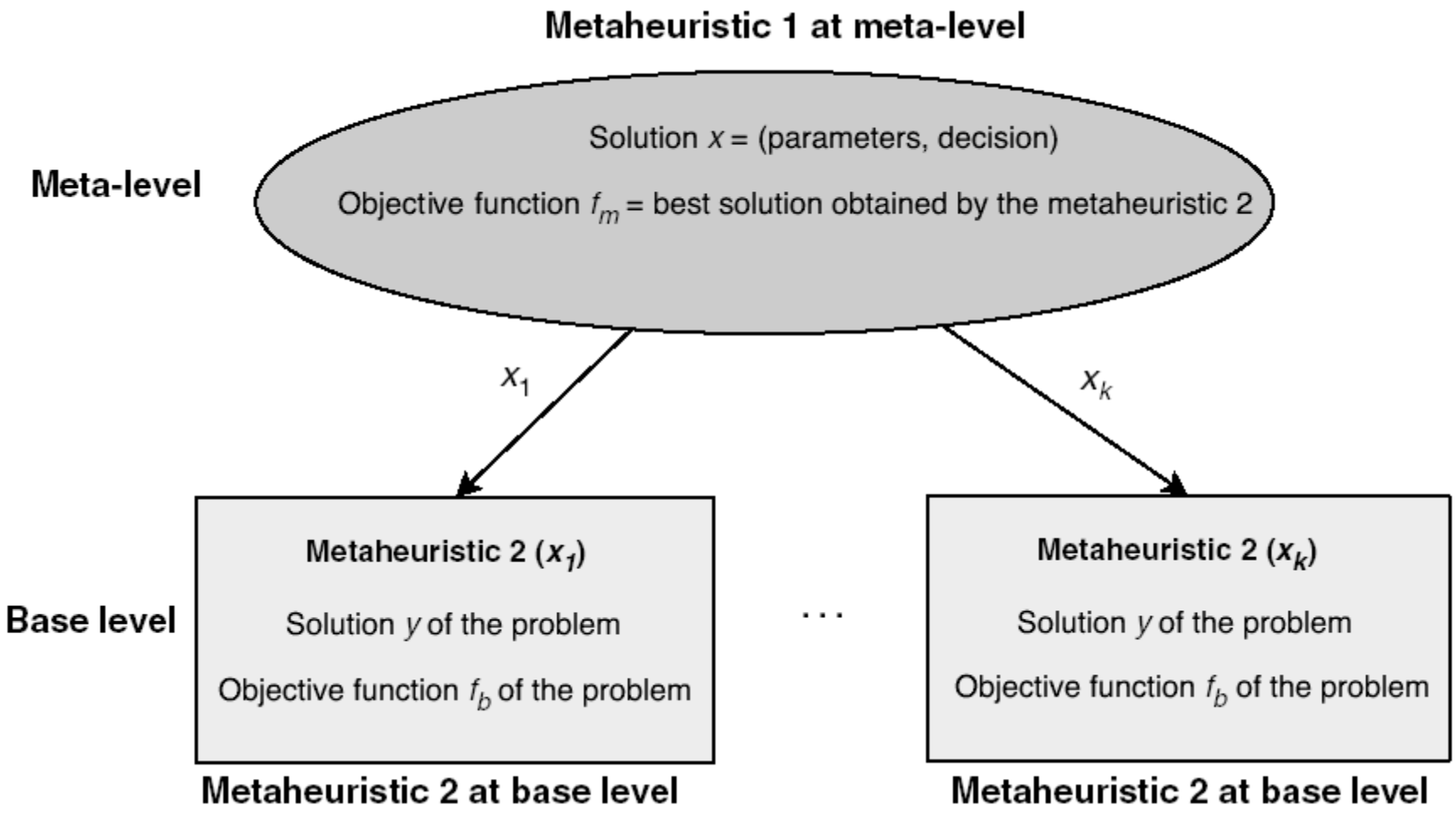
- However, a small number of experiments may be performed by using:
  - **Latin hypercube designs**
  - **Sequential design**
  - **Fractional design**
- Other approaches used in **machine learning** community:
  - **Racing algorithms**

Meta-optimization Approach

Meta-optimization Approach

- In off-line parameter initialization, the search for the best tuning of parameters of a metaheuristic in solving a given problem may be formulated as an optimization problem.
- This **meta-optimization approach** may be performed by any (meta)heuristic, leading to a meta-metaheuristic (or meta-algorithm) approach.
- Meta-optimization may be considered a hybrid scheme in metaheuristic design.

Meta-optimization using a meta-metaheuristic



Meta-optimization Approach

- This approach is composed of two levels:
  - the meta-level
  - the base level
- At the meta-level, a metaheuristic operates on solutions (or populations) representing the parameters of the metaheuristic to optimize.
- For instance, it has been used to optimize:
  - Simulated annealing by GA
  - Ant colonies by GA
  - A GA by a GA

Meta-optimization Approach

- A solution x at the meta-level will represent all the parameters the user wants to optimize:
  - Parameter values
    - such as the size of the tabu list for tabu search, the cooling schedule in simulated annealing, the mutation and crossover probabilities for an evolutionary algorithm
  - Search operators
    - Such as the type of selection strategy in evolutionary algorithms, the type of neighborhood in local search, and so on.

Meta-optimization Approach

- At the meta-level, the objective function f_m associated with a solution x is generally the best found solution (or any performance indicator) by the metaheuristic using the parameters specified by the solution x.
- Hence, to each solution x of the meta-level will correspond an independent metaheuristic in the base level.
- The metaheuristic of the base level operates on solutions (or populations) that encode solutions of the original optimization problem.

Meta-optimization Approach

- The objective function f_b used by the metaheuristic of the base level is associated with the target problem.
- Then, the following formula holds:

$$f_m(x) = f_b(\mathrm{Meta}(x))$$

  - where Meta(x) represents the best solution returned by the metaheuristic using the parameters x.

Online Parameter Initialization

Online Parameter Initialization

- The drawback of the off-line approaches is their high computational cost, particularly if this approach is used for each input instance of the problem.
- Indeed, the optimal values of the parameters depend on the problem at hand and even on the various instances to solve.
- Then, to improve the effectiveness and the robustness of off-line approaches, they must be applied to any instance (or class of instances) of a given problem.

Online Parameter Initialization

- Another important drawback of off-line strategies is that the effectiveness of a parameter setting may change during the search; that is, at different moments of the search different optimal values are associated with a given parameter.
- Hence, online approaches that change the parameter values during the search must be designed.

Online Parameter Initialization

- Online approaches may be classified as follows:
  - **Dynamic update**
    - In a dynamic update, the change of the parameter value is performed without taking into account the search progress.
    - A random or deterministic update of the parameter values is performed.
  - **Adaptive update**
    - The adaptive approach changes the values according to the search progress.
    - This is performed using the memory of the search.

Online Parameter Initialization

- A subclass of adaptive, referred to as **self-adaptive approach**, consists in “evolving” the parameters during the search.
- Hence, the parameters are encoded into the representation and are subject to change as the solutions of the problem.

Step 13. Analyze the Performance of Algorithm

Analyze the Performance of Algorithm

- After parameters tuning we must obtain the experimental results for different indicators
- Methods from statistical analysis can be used to conduct the performance assessment of the designed metaheuristics.
- For nondeterministic (or stochastic) algorithms, many trials (at least 10, more than 100 if possible) must be carried out to derive significant statistical results.

Analyze the Performance of Algorithm

- From this set of trials, many measures may be computed:
  - mean,
  - median,
  - minimum,
  - maximum,
  - standard deviation,
  - the success rate that the reference solution (e.g., global optimum, best known, given goal) has been attained, and so on.

Analyze the Performance of Algorithm

- The success rate represents the number of successful runs over the number of trials.

$$\text{success rate} = \frac{\text{number of successful runs}}{\text{total number of runs}}$$

Step 14. Report Result

Report Result

- The interpretation of the results must be explicit and driven using the defined goals and considered performance measures.
- In general, it is not sufficient to present the large amount of data results using tables.
- Some **visualization tools** to analyze the data are welcome to complement the numerical results.

Report Result

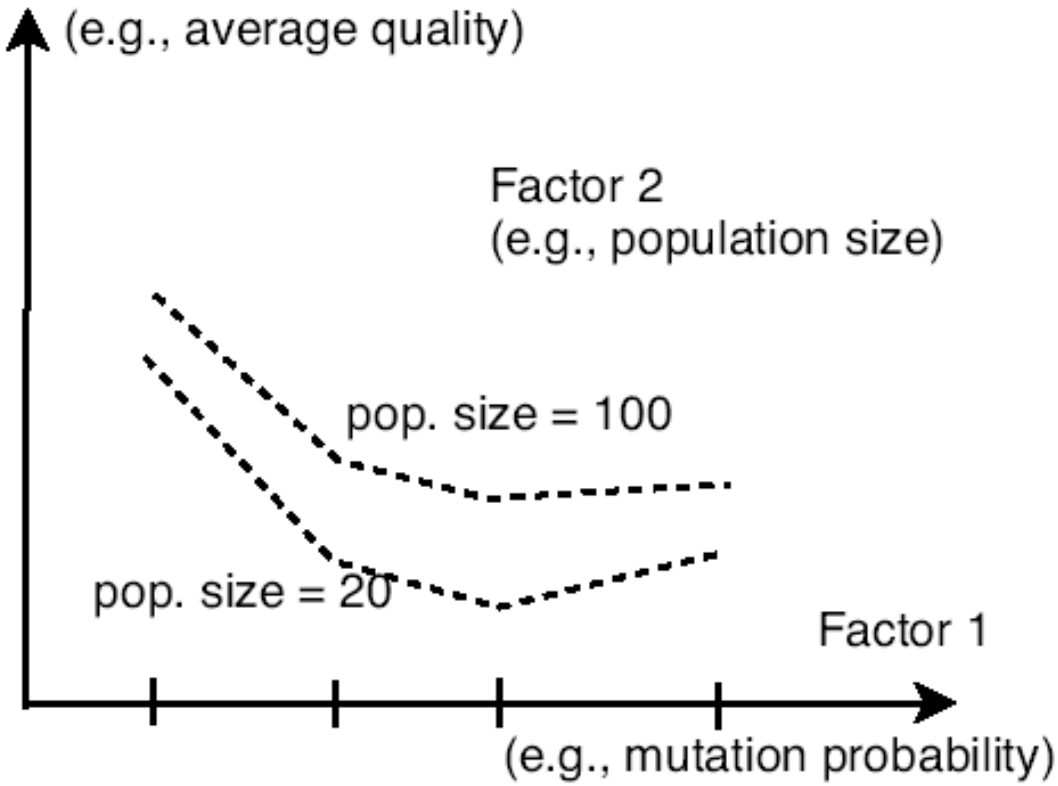
- Graphical tools allow a better understanding of the performance assessment of the obtained results, such as
  - Interaction plots
  - Scatter plots
  - Box plots

Report Result

- **Interaction plots**
  - represent the interaction between different factors and their effect on the obtained response (performance measure)
- **Scatter plots**
  - to illustrate the compromise between various performance indicators.
  - For instance, the plots display quality of solutions versus time, or time versus robustness, or robustness versus quality

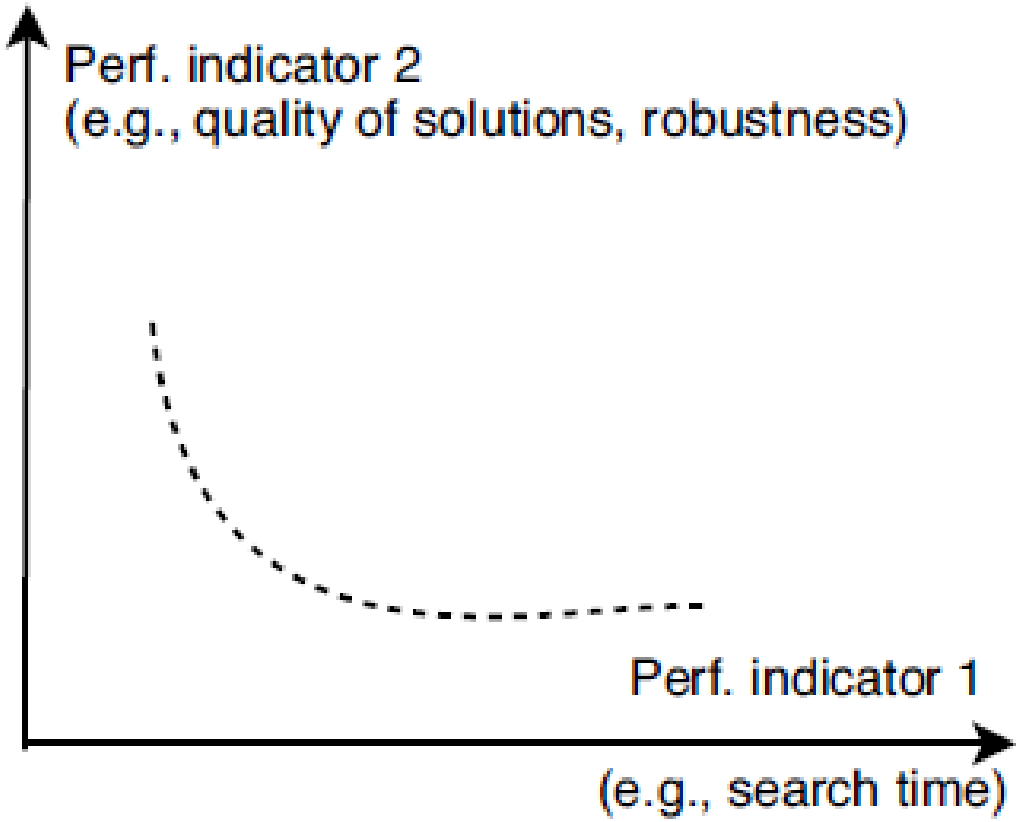
Report Result

- **Example:** Interaction plot: Interaction plot analyzes the effect of two factors (parameters, e.g., mutation probability, population size in evolutionary algorithms) on the obtained results (e.g., solution quality, time).



Report Result

- **Example: Scatter plot:** The scatter plot analyzes the trade-off between the different performance indicators (e.g., quality of solutions, search time, robustness).

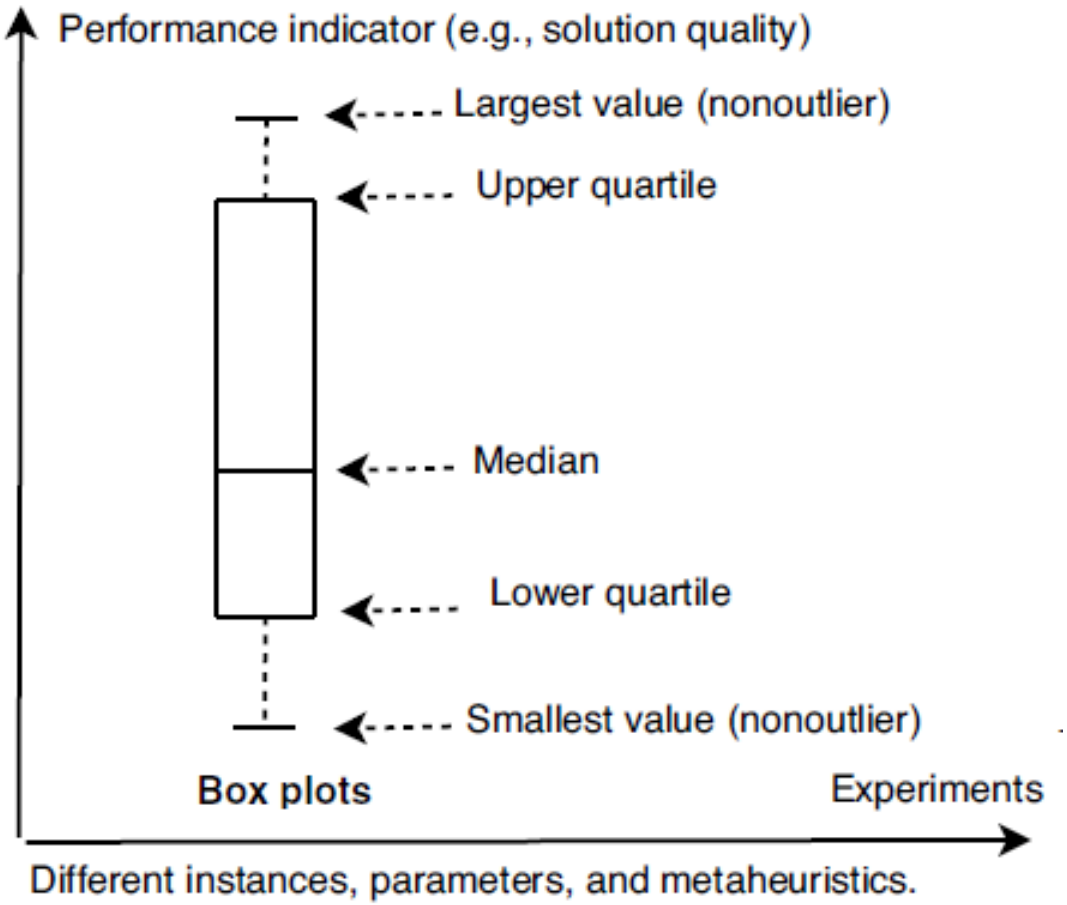


Report Result

- **Box plots**
  - illustrate the distribution of the results through their five-number summaries:
    - the smallest value,
    - lower quartile (Q1),
    - median (Q2),
    - upper quartile (Q3), and
    - largest value
  - They are useful in detecting outliers and indicating the dispersion and the skewness of the output data without any assumptions on the statistical distribution of the data.

Report Result

- Example: Box plot



References

References

- El-Ghazali Talbi, **Metaheuristics : From Design to Implementation**, John Wiley & Sons, 2009.

The End